

REMARKS

Claims 1-20 are amended. No new matter is added by these amendments. Claims 1-20 are pending. By amending the claims, applicant is not conceding that the claims are unpatentable over the art cited by the Office Action and is not conceding that the claims are non-statutory under 35 U.S.C. 101, 102, and 103 as the claim amendments are only for the purpose of facilitating expeditious prosecution. Applicant respectfully reserves the right to pursue the subject matter of the claims as it existed prior to any amendment, and other claims, in one or more continuation and/or divisional applications. Applicant respectfully requests reconsideration and allowance of all claims in view of the amendments above and the remarks that follow.

Rejections under 35 U.S.C. 101

Claims 9-12 are rejected under 35 U.S.C. 101 because "A signal, a form of energy, does not fall within one of the four statutory classes of § 101." Claims 9-12 are amended to recite a storage medium, which is a tangible physical article or object and statutory under 35 U.S.C. 101.

Rejections under 35 U.S.C. 102 and 103

Claims 1-3, 5, 6, 8-15, and 17-19 are rejected under 35 U.S.C. 102(e) as unpatentable over Koderia (US Patent 6,718,484 B1). Claims 4, 7, 16, and 20 are rejected under 35 U.S.C. 103(a) as unpatentable over Alverson (US Patent 6,848,097 B1). Applicant respectfully submits that the claims are patentable over Koderia and Alverson because Koderia and Alverson do not teach or suggest all elements of the claims for the reasons argued below.

Claim 1 recites: "if the identifier was saved in response to the thread that executes the instance of the program encountering the entry breakpoint and the scoped breakpoint was encountered by the thread, halting execution of the thread that encountered the

scoped breakpoint," which is not taught or suggested by Koderia and Alverson for the reasons argued below.

In contrast to claim 1, Koderia at column 12, lines 24-25 describes that "In subsequent step 5, a stopped thread is brought into a suspended state," and Koderia makes the bringing of the stopped thread into the suspended state conditional on the following three Koderia judgments:

1) "When it is judged in step 1 that the thread stoppage is caused by presence of a breakpoint" (Koderia at column 11, lines 44-45);

2) "When it is judged that the barrier synchronization is not currently being performed, on the basis of a state in which a NULL pointer is registered in the enabled-barrier synchronization-point management block" (Koderia at column 12, lines 1-4); and

3) "when it is judged in step 3 that the barrier synchronization-point flag of the breakpoint management block 611-i that manages a breakpoint having caused stoppage is in an ON state" (Koderia at column 12, lines 15-18).

All three of the aforementioned Koderia judgments are unrelated to, and do not teach or suggest, "if the identifier was saved in response to the thread that executes the instance of the program encountering the entry breakpoint and the scoped breakpoint was encountered by the thread," for the reasons argued below.

With respect to the first Koderia judgment, when Koderia performs the "stopped thread is brought into a suspended state" at column 12, lines 24-25, the Koderia stopped thread is still stopped at the same breakpoint whose "presence" "caused" "the thread stoppage" (Koderia at column 11, lines 44-45), so Koderia does not teach or suggest both the encountering of the entry breakpoint and the encountering of the scoped breakpoint recited in claim 1 because the first Koderia judgment only judges one breakpoint.

With respect to the second Koderia judgment, Koderia at column 11, lines 63-66 "writes a NULL pointer into the enabled-barrier synchronization-point management

block 612” “when the multithreaded program 20 is started, as a part of the initialization processing.” Thus, the second Koder judgment is true the first time that Koder encounters a breakpoint after the multithreaded program 20 is started. Hence, in order for Koder to bring the “stopped thread” “into a suspended state” at column 12, lines 24-25, only one breakpoint has been encountered, so Koder does not teach or suggest both the encountering of the entry breakpoint and the encountering of the scoped breakpoint recited in claim 1 because the second Koder judgment is true on the presence of a first Koder breakpoint.

With respect to the third Koder judgment, the Koder “barrier synchronization-point flag” which is judged by the third Koder judgment (at column 12, lines 15-18) is set at Koder column 11, lines 6-9: “In subsequent step 7, a barrier synchronization-point flag of the breakpoint management block 611-i which manages the selected breakpoint is brought into an ON state,” and this setting of the barrier synchronization-point flag occurs prior to the “[start of] the multithread program 20,” which does not occur until column 11, lines 28-30. Thus, the third Koder judgment is unrelated to, and does not teach or suggest both the encountering of the entry breakpoint and the encountering of the scoped breakpoint recited in claim 1 because the third Koder judgment checks a flag that was set prior to the start of the execution of the multithread program that encountered the breakpoint, so the Koder “barrier synchronization-point flag” is unrelated to the encountering of any breakpoint, much less the encountering of the entry breakpoint and the scoped breakpoint of claim 1.

Thus, Koder does not teach or suggest “if the identifier was saved in response to the thread that executes the instance of the program encountering the entry breakpoint and the scoped breakpoint was encountered by the thread, halting execution of the thread that encountered the scoped breakpoint,” as recited in claim 1.

In contrast to claim 1, Alverson at column 21, lines 30-38 recites “If the current instruction is instead a breakpoint, the routine continues to step 1120, where attempted execution of a BREAK instruction will cause a privileged instruction exception to be

raised, thus transferring flow of control to the privileged instruction exception handler for this thread which in turn invokes the breakpoint handler for the thread. Thus, the routine continues to step 1125 where execution of the target instructions are halted." Thus, Alverson merely halts the routine if a break instruction is attempted to be executed, so Alverson also does not teach or suggest "if the identifier was saved in response to the thread that executes the instance of the program encountering the entry breakpoint and the scoped breakpoint was encountered by the thread, halting execution of the thread that encountered the scoped breakpoint," as recited in claim 1.

Claim 1 further recites: "if the identifier was not saved, the thread that executes the instance of the program did not encounter the entry breakpoint, and the scoped breakpoint was encountered by the thread that executes the instance of the program, allowing execution of the thread to continue after the scoped breakpoint was encountered without giving control to a user," which is not taught or suggested by Koderia and Alverson for the reasons argued below.

In contrast to claim 1, Koderia at column 7, lines 10-18 recites: "When a breakpoint which does not belong to the barrier synchronization point is set, the execution means 18 operates as follows. Specifically, when after barrier synchronization operation is started in response to stoppage of a task at a breakpoint belonging to a certain barrier synchronization point, a task is stopped at a certain breakpoint not belonging to the barrier synchronization point, the execution means 18 resumes the multitask program 10 while ignoring the certain breakpoint."

Koderia judges whether a breakpoint "does not belong to the barrier synchronization point" by judging "whether the breakpoint management block 611-i is connected to the chain from the enabled-barrier synchronization-point management block 612," as recited by Koderia at column 12, lines 59-62.

A Koderia breakpoint management block 611-i is connected to the chain in Koderia at column 11, lines 11-15: "In subsequent step 8, the breakpoint management block 611-i which manages the selected breakpoint is connected to the chain from the constituent

breakpoint management chain of the created barrier synchronization-point management block 610-i," which occurs after breakpoints are set (block ST6 of Fig. 7) and prior to the start of the multithreaded program at Koder a column 11, lines 29-31. Thus, the Koder a breakpoint is connected to the chain prior to the start of the Koder a multithreaded program, so whether or not a Koder a breakpoint is connected to or belongs to a chain is unrelated to the Koder a breakpoint being or not being encountered.

Thus, when Koder a at column 7, lines 10-13 determines whether "a breakpoint ... does not belong to the barrier synchronization point," Koder a is making this determination based on data that was set prior to the Koder a multithreaded program starting, so Koder a's judgments at column 7, lines 10-19 are unrelated to encountering breakpoints, so Koder a does not teach or suggest "if the identifier was not saved, the thread that executes the instance of the program did not encounter the entry breakpoint, and the scoped breakpoint was encountered by the thread that executes the instance of the program, allowing execution of the thread to continue after the scoped breakpoint was encountered without giving control to a user," as recited in claim 1.

Alverson also does not teach or suggest "if the identifier was not saved, the thread that executes the instance of the program did not encounter the entry breakpoint, and the scoped breakpoint was encountered by the thread that executes the instance of the program, allowing execution of the thread to continue after the scoped breakpoint was encountered without giving control to a user," as recited in claim 1 because Alverson merely halts a routine if a break instruction is attempted to be executed, as described by Alverson at column 21, lines 30-38: "If the current instruction is instead a breakpoint, the routine continues to step 1120, where attempted execution of a BREAK instruction will cause a privileged instruction exception to be raised, thus transferring flow of control to the privileged instruction exception handler for this thread which in turn invokes the breakpoint handler for the thread. Thus, the routine continues to step 1125 where execution of the target instructions are halted."

Thus, Koder and Alverson, alone or in combination, do not teach or suggest all elements of claim 1. Claims 5, 9, 13, and 17 include similar elements as argued above for claim 1 and are patentable over Koder and Alverson for similar reasons as those argued above. Claims 2-4, 6-8, 10-12, 14-16, and 18-20 are dependent on claims 1, 5, 9, 13, and 17, respectively, and are patentable for the reasons argued above, plus the elements in the claims.

RECEIVED
CENTRAL FAX CENTER

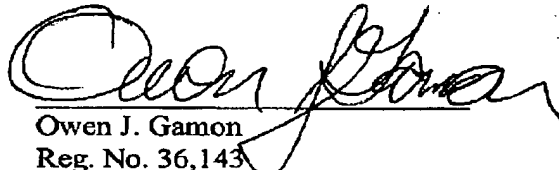
MAR 13 2008

Conclusion

Applicant respectfully submits that the claims are in condition for allowance and notification to that effect is requested. The Examiner is invited to telephone Applicant's attorney (651-645-7135) to facilitate prosecution of this application.

If necessary, please charge any additional fees or credit overpayment to Deposit Account No. 09-0465.

Respectfully submitted,

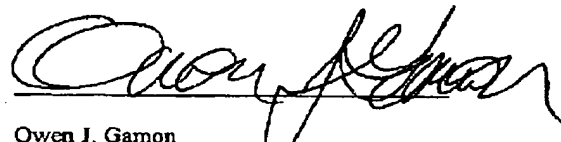

Owen J. Gamon
Reg. No. 36,143
(651) 645-7135

Date: March 13, 2008

IBM Corporation
Intellectual Property Law
Dept. 917, Bldg. 006-1
3605 Highway 52 North
Rochester, MN 55901

CERTIFICATE UNDER 37 C.F.R. 1.8

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to Mail Stop Amendment, Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450, or is being transmitted via facsimile to the U.S. Patent and Trademark Office, 571-273-8300, on: March 13, 2008.


Owen J. Gamon
Registration No. 36,143